




CSILLING BÁLINT

PROGRAMOZÓI DOKUMENTÁCIÓ

L02-R4B 
 E-mail-cím 
 LC3M9F 

TARTALOMJEGYZÉK

1. MEGVALÓSÍTOTT FELADAT	2
2. PONTOSÍTOTT SPECIFIKÁCIÓ	2
2.1 ALAPVETŐ FUNKCIÓK	2
2.2 MEGJELENÍTÉS.....	2
2.3. A MEGVALÓSÍTANDÓ MŰKÖDÉS	2
2.4. ELVÁRT FÁJLFORMÁTUM.....	3
2.5. HIBAKEZELÉS	3
3.TERV.....	3
3.1 ADATSZERKEZET ÉS OSZTÁLYOK KAPCSOLATA	3
3.2 UML DIAGRAM ÉS OSZTÁLYOK KAPCSOLATA	4
3.3 AUTO-KOKTÉL ALGORITMUSA.....	4
3.4 MENÜPONTOK ÉS MŰKÖDÉS	5
4. IMPLEMENTÁCIÓ.....	5
4.1 ADATSZERKEZET.....	5
4.2 ALGORITMUSOK.....	6
4.3 MENÜRENDSZER.....	7
5. BŐVÍTHETŐSÉG.....	9
6. TESZTELÉS.....	9
6.1 GTEST LITE.....	9
6.3 JPORTA COVERAGE.....	9
6.4 MEMÓRIA HASZNÁLAT	9
7. ÖN REVÍZIÓ ÉS MÓDOSÍTÁSOK.....	10
8. FELHASZNÁLT KÖNYVTÁR.....	10
8.1 ECONIO	10
9. A PROGRAM RÉSZEI	10

1. MEGVALÓSÍTOTT FELADAT

Az program egy C++ nyelven megvalósított nyilvántartás szerű rendszer, amely képes koktél receptek és alapanyagok eltárolására és kezelésére.

2. PONTOSÍTOTT SPECIFIKÁCIÓ

2.1 Alapvető funkciók

A szoftver fő feladata, hogy koktélok receptjeit eltárolja, lehetőséget biztosítva új receptek felvételére, megjelenítésére, esetleg törlésére. Lehetősége van a felhasználónak eltárolni egy fájlban a recepteket, illetve beolvasni őket. A program képes ezen kívül alapanyagok eltárolására is, illetve ezekből megállapítani, hogy milyen koktélok lehetne előállítani az adott alapanyagokból.

2.2 Megjelenítés

Konzolos | Interaktív

A program egy konzolos felületen keresztül lehet kezelni. A program a menüpontokban mindig egymás alatt kiírja az abban az almenüben elérhető menüpont nevét egy kis dobozban, illetve egy dobozban egy X jelenik meg, hogyha éppen ezen a menüponton van a felhasználó. Navigálni az „s” és „w” billentyűkkel lehet, ha a kívánt menüponton vagyunk az enter gomb megnyomásával jutunk tovább. Minden almenüben a legutolsó gomb a vissza gomb, amivel a fő menübe jutunk vissza. A programnak 3 rétegű menürendszere van. Minden alkalommal amikor a felhasználó egy operációt végrehajtott, gyakorlatilag bármilyen program funkciót, ezután a főmenübe fog menni.

2.3. A megvalósítandó működés

- Indításkor mindig a főmenü jelenik meg.
- Bármelyik menüpontból van visszalépési lehetőség, ami a főmenübe visz.
- Minden cselekmény után a főmenübe navigál vissza a program.
- Csak akkor történik képfrissítés, hogyha felhasználói esemény történik.
- Adott formátumú fájl beolvasása
- Alapanyag vagy ital törlése
- Automatikus mentés nincs a programnak
- Alapanyagok tulajdonságait meg lehet adni, amit eltárol a program.
- A program az alapanyagok alapján képes kiszámolni a koktél összesített tulajdonságait, pl. alkoholszázalék.
- Listázásnál a koktélok összes alapanyagát kilistázza, illetve kulcsfontosságú adatokat, allergén-e, összesített alkoholszázalék, édességi szint (édes alapanyagok száma), összmennyiség.
- Alapanyag hozzáadásnál csak nevet kell megadni egy intuitív felületen
- Koktél hozzáadásnál meg kell adni előre a koktél alapanyagainak számát, majd egyesével az alapanyag típusát, és összes többi tulajdonságát, mindig azt, amit kér a program, minden bevétel végén enter kell nyomni.
- A listázás után enter gomb visszajut a főmenübe.

2.4. Elvárt fájlformátum

A program csak .dat és .txt kiterjesztésű fájlokat képes kezelni.

A koktélok az alábbi formátumban kell, hogy szerepeljenek:

A koktél neve

6Liquor

Gin

100

szaraz

4.5

true

5Mixer

Tonic

300

true

true

false

&

...új név

Egy koktél neve után az alapanyagok következnek, ha nincs több alapanyag azt egy & jelzi. Az alapanyagoknak 3 fajtája lehet a koktéلكészítésnél, Liquor, Mixer, Garnishes. A fájlban ezt a név beírásával kell jelezni, megelőzve a szó hosszával.

6Liquor	5Mixer	9Garnishes
Alapanyag megnevezése		
Mennyisége milliliterben/ db		
Liquor típusa: száraz	Edesség:true	Citrusos: true
Alkoholszázalék: 14.5	Szénsavas:true	Csipos: false
Edesség: true	Allergén:true	Allergen: false

2.5. Hibakezelés

A menürendszerben a w-s gombok nem hatásosak, hogyha túlmennénk a legelső vagy legutolsó menüponton. Más gombokra egyébként sem reagál. (Triviális kivételektől eltekintve, melyek az operációs rendszerhez szólnak pl. Alt+F4, Ctrl+C stb...)

Nem racionális adatbevitel esetén figyelmeztet a program egy szöveggel, és egyszer újból lehet próbálni a bevitelt. (Például 120%-os alkoholszázalék, több száz literes koktél)

Hibásan megadott fájlnevnél nem fog történni beolvasás, felhasználó értesítés.

Ha nem sikerül megnyitni a fájl, szintén nem fog fájlba írás/ beolvasás történni, felhasználó értesítés.

3. TERV

3.1 Adatszerkezet és osztályok kapcsolata

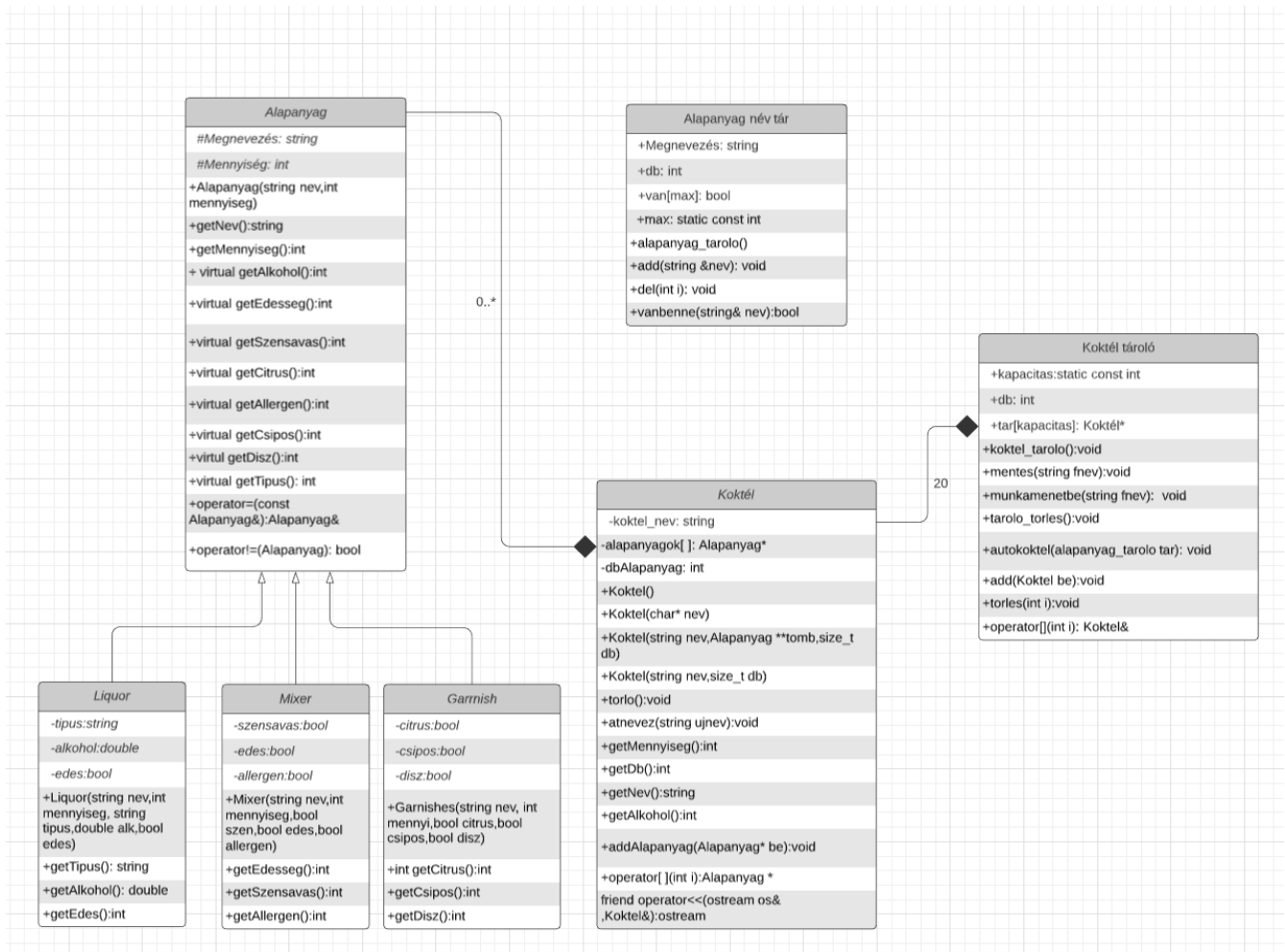
A rendszerben szükség van alapanyag nevekre, alapanyagok, koktélok tárolására. Az összefüggéseket az UML diagram ábrázolja. Lásd 4. oldal.

Egy **koktél** egy olyan osztály fog tárolni, mely rendelkezik egy **alapanyag pointer tömbbel**. Egy alapanyag egy **őosztály**, amelynek a koktéلكészítés értelmében 3 típusa lehet: Liquor, Mixer és Garnishes. Ezek képzik a **leszármazott osztályokat**. Az alapanyag számos **virtuális getter** függvénnyel rendelkezik, így képes betekintést adni az alapanyagok tulajdonságaira, amelyeknek a nevei beszédesek. Egy koktélnak önmagában tulajdonképpen **csak neve van**, minden más ez alapján lesz kiszámolva, eltárolni fölösleges.

Egy említett koktél a fix méretű **koktél tároló osztályban** van, ez képes átláthatóan kezelni a fájlműveleteket és az egész adatbázisra belátást igénylő +autokoktél() függvényt. A koktéltárolóban az összes koktél és alapanyag **dinamikusan** van tárolva.

Az alapanyag tároló egy olyan osztály, amely képes eltárolni **alapanyagok neveit**, erre csak azért lehet szükség, hogy megtudjuk a koktél tárolóból milyen koktél elkészítéséhez van elegendő alapanyagunk. Ezért **csak az alapanyagok nevére** van szükségünk, ezt is egy fix méretű tömbben tároljuk.

3.2 UML diagram és osztályok kapcsolata



3.3 Auto-Kocktél algoritmus

Az algoritmust megvalósító függvény a Kocktél tároló osztályban van, bemeneti paramétere egy alapanyag tároló, ez teszi ki az elérhető alapanyagokat, ezekből próbál majd lehetséges kocktélok keresni, ahol minden szükséges alapanyag rendelkezésre áll a tárolóban. Az egyszerű algoritmus tulajdonképpen örfeltételes lineáris keresések egymásba ágyazva.

Pszedukód: (Python 3.10)

```

for koktel in osszes_koktel:
    lehetséges=True

    for alapanyag in koktel.alapanyagok:
        benne_van=False

        for tarolo_alapanyag in alapanyag_tarolo:
            if tarolo_alapanyag == alapanyag:
                benne_van=True
                break;

        if benne_van==False:
            lehetséges= False
            break;

    if lehetséges:
        /*Konzolra írás*/

```

3.4 Menüpontok és működés

A koktélok menüpontban van lehetőségünk a rendszerben lévő koktélok kiírni, koktélt hozzáadni, törölni és visszamenni a fő menübe.

Az alapanyagok menüpontban lehetőségünk van alapanyagok listázására, hozzáadásra, törlésére és visszamenni.

Az Auto-Koktel (röviden *Auto-kok*) egy viszonylag egyszerű algoritmus mely képes megmondani milyen koktélok készíthetők el a jelenlegi alapanyagokból.

Az operációk menüpontban lehet beolvasni és kiírni a koktél adatbázist. Mindkét esetben csak a fájl nevét kell megadni kiterjesztéssel és entert nyomni.



4. IMPLEMENTÁCIÓ

4.1 Adatszerkezet

A feladat az volt, hogy egy koktélt el lehessen tárolni, erre van a koktél osztály. Egy koktélnak a korlátlan számú alapanyagát egy Alapanyag pointer tömb tárolja, ez egy heterogén kollekció. Az alapanyagoknak 3 típusa lehet, magyarra fordítva likőr, kísérő és aromák. Mindhárom osztálynak közös tulajdonsága, hogy van neve és mennyisége, a többi az osztály specifikus, mint alkoholszázalék, édesség stb. Az alapanyagok őssztálynak számos triviális virtuális getter függvénye van.

A koktél tároló osztályban tárolódnak a koktélok, pontosabban a dinamikusan foglalt koktélok pointerre. Úgy van megvalósítva, hogy a statikus kapacitása van, és amikor létrehozuk, lefoglal a kapacitásnak megfelelő koktélt, „alap” névvel, és ha hozzáadunk egy koktélt akkor ezt módosítja, illetve, hogy a tároló darabszám változót. Egy Koktél 24 byte memória, és alapértelmezetten 20 koktél tárolható el. Ha töröljük, felszabadítja a dinamikusan foglalt alapanyagokat, alapanyag pointer tömböt és magát a koktélt, és a helyére egy alapértelmezett alap koktélt rak, majd átrendezi a tároló kezdve a törlés indexével, hogy a nem alap koktélok legyenek elől. Nincs hatékonyabb rendezés véleményem szerint, nem érdemes lecserélni másra. Azért nem bináris fában vannak tárolva, mert sokszor újra kéne építeni a fát, illetve nem használunk keresést a tárolóban, ezért minden szempontból hatékonyabb ez az adatszerkezet. Fontos kiemelni, hogy a koktél tároló nem egy isteni objektum, nem lát mindenre rá, azonban egyszerűsége miatt nem privát. Kódkomplikációval privát is lehetne.

Az alapanyag tároló csupán alapanyagok nevét tárolja, ezt tekintjük egy olyan konténernek, amibe berakunk alapanyagokat, hogy aztán megmondhassuk, hogy milyen koktélok készíthetők el. Egy alapanyag annak ellenére, hogy más mennyisége van például, az nem jelenti azt, hogy nem azonos, emiatt semmilyen más tulajdonságot nem kell tárolni a nevükön kívül. Erre szolgál az alapanyag

tároló, nem dinamikusan tárol kapacitásnyi stringet, és egy foglaltsági tömböt. Hogy ne kelljen gyakran rendezni, így, ha valamilyen alapanyag felszabadul, a foglaltsági mátrixban false lesz a foglaltsága, is így fölülírható lesz.

4.2 Algoritmusok

Az Auto-Koktél algoritmus magyarázata: [3.3 Auto-Koktél algoritmus](#)

A programban a beolvasás és mentés funkciójának lényege, hogy a koktél tároló tartalmát maradéktalanul kiírják egy paraméterként kapott fájlba, vagy éppen onnan olvassák be. Nagyon fontos volt, hogy a két funkció képes legyen egymás közt felhasználni a fájlokat, magyarul amit az egyik elkészít, a másik tudja használni. A munkamenetbe nevű függvény a paraméterként kapott fájlt megnyitja, és egy ciklus indul el, ami mindig egy koktél nevét fogja várni, majd egy alapanyag típusát és az alapanyag specifikus tulajdonságok beolvasása után a koktélhoz hozzáadásra kerül egy dinamikusan foglalt alapanyag, ha a program egy & jelet lát, ez azt jelzi, hogy a koktélnek nincs több alapanyag, megkezd egy új koktél beolvasását, miután a mostanit berakta a koktél tárolóba. Ha részletesen szeretné látni a fájlformátumot, illetve a beolvasás formátumát [navigáljon ide](#).

A beolvasásnál létre kell hozni számos string változót, erre azért van szükség, mert ezekbe bármilyen adattípust bele lehet rakni, így még a boolean típusokat is be lehet ezekbe olvasni, és ezt majd átalakítva megkaphatja az alapanyag konstruktora.

```
void koktel_tarolo::munkamenetbe(const string& fnev) {
    std::ifstream is;           // input file stream, vagyis olvasunk egy fájlból

    is.open(fnev);              // megnyitjuk a fájlt
    if (is.fail()) {
        std::cout << "Nincs ilyen fájl!";
        return;
    }

    /* Az eddigi tárolót maradéktalanul töröljük. */
    koktel_torles();

    /* Új tároló inicializálás */
    for (auto & i : tar) {
        i=new Koktel("Alap",0);
    }

    /*Kezdetben nincs egy koktél sem*/
    db=0;

    /* Alapanyag mennyisége, neve, első, második harmadik tulajdonsága.
     * Többféle lehet az utolsó 3. */
    string mennyisege,nev,egyes, ketto,harom;
```

A ... azt jelzi, hogy a többi alapanyag típus is pont így van megvalósítva, csak a saját konstruktorral.

```
while (std::getline(is, s)) {
    /* Beolvasott adat a koktél neve, átnevezzük az alap koktélt erre. */
    tar[db]-->atnevez(s);
    /*Az s itt most egy alapanyag típusának neve, értelemsszerűen amit olvasott annak megfelelően
    készíti el az alapanyagot. */
    std::getline(is, s);
    while(s!="&"){
        if (s=="6Liquor"){
            std::getline(is, nev);
            std::getline(is, mennyiség);
            std::getline(is, egyes);
            std::getline(is, ketto);
            std::getline(is, harom);

            /* Lefoglalás és hozzáadjuk a koktélhoz az alapanyagot. */
            auto *id= new Liquor(nev,std::stoi(mennyiség),egyes,std::stoi(ketto),stob(harom));
            tar[db]-->addAlapanyag(id);
        }
        ...
        std::getline(is, s);
    }
    /* Egy koktél elkészült */
    db++;
}
is.close();
}
```

4.3 Menürendszer

A menürendszer a programban véleményem szerint remekül van implementálva. Mindent a menurendszer.cpp fájl tartalmaz. A működéshez 3 segédfüggvény van létrehozva.

Az egyik függvény: box egy paraméterként kapott stringet ír ki, és a szintén paraméterként kapott bool érték alapján egy X-et rajzol, vagy sem, de mind ezt egy ASCII dobozban teszi meg.

A menü függvénynek egy string tömb és egy szám paramétere van, kiírja az összes menüpontot a box függvénnyel a szabványos kimenetre, és csak az a menüpont lesz aktív, azaz egy X lesz a dobozban, amelyiknek a sorszáma megegyezik a paraméterként kapott sorszámmal.

```
void menu(int index, string *tomb) {
    bool check = false;
    for (int i = 0; !tomb[i].empty(); ++i) {
        if (i == index - 1) {
            check = true;
        }
        box(tomb[i], check);
        check = false;
    }
}
```

A navigáció függvény paraméterként kapja a teljes menü kétdimenziós string tömbjét, és figyelni fogja a billentyűzetet. A felhasználó s és w gombokkal fel le tud lépni a menüben, ha ezeket a gombokat nyomja meg, akkor újra meghívja a menü függvényét, de most az aktív menüpont eggyel lejjebb vagy feljebb lesz. A navigációs függvényben van egy belső változó, ami alapértelmezetten 0, tehát a 0. menüpont aktív, és a billentyűkkel ezt lehet növelni. Természetesen nem lehet több mint a menüpontok száma vagy kevesebb mint 0. Ha a felhasználó enter nyom valamelyik menüponton, ezt azt jelenti, hogy kiválasztotta a menüpontot, a jelenlegi belső változó a függvény visszatérési értéke.

```

int navigation(string *tomb) {
    econio_rawmode(); /* Szüksége van a getch függvénynek a működéséhez. */
    int x = 1; /* Alapértelmezetten az első menüpont aktív. */
    int meret; /* Hány menüpont van összesen? */
    int key; /* Melyik gombot nyomta meg a felhasználó? */

    #ifndef SIM /* Jportán nem kell rajzolni. */
    menu(x, tomb);
    #endif

    for (meret = 0; !tomb[meret].empty(); ++meret) { /* Hany eleme a tomb? */ }

    while (true) {
        #ifndef SIM /* Jportán nem kell rajzolni. */
        econio_clrscr();
        menu(x, tomb);
        #endif

        //key = econio_getch(); /* Bekérjük az inputot? w vagy s */
        key = econio_getch();
        if (key == 119 && x > 1) /*Ha s betűt nyomott lelép a következőre kivéve ha ez a legalsó
        menüpont. */
            x--;
        if (key == 115 && x < meret) /*Ha w betűt nyomott fellép a következőre kivéve ha ez a legfelső
        menüpont. */
            x++;
        if (key == 10) /* Ha enter nyom a felhasználó kilépés. */
            break;

    }
    if (tomb[4] == "KILEPES" && x == meret)
        return -1; /*KILEPESRE KATTINTOTT*/
    if (x == meret)
        return 0; /*A VISSZA GOMBRA KATTINTOTT*/

    return x;
}

```

A menürendszer alapja tehát egy 2 dimenziós string tömb, amiben a menü és almenü pontjai vannak benne. Mivel a menürendszer kétdimenziós, ezért kétszer meg lesz hívva a navigációs függvény, azaz megtudjuk, hogy a felhasználó a fő menüben mire ment, majd a második navigációs függvény, ami már az előző almenü indexével tér vissza, meg fogja mondani, hogy az almenüben mire kattintott. Ezt a két számot eltároljuk és egy egymásba ágyazott switchben minden lehetséges kombináció meg van valósítva. Például: 1->1 azt jelentené, hogy a főmenüben az első, az almenüben a második menüpontra kattintott, ami egyébként a koktél hozzáadása lenne. Ha kilépésre megy, program kilép mert a navigáció -1-et fog visszaadni. Ha 0-át ad vissza a navigáció, a program visszamegy a főmenübe.

```

/*A menüpontok tömbje, ezeket írja ki. Lehet módosítani a neveket. */
string tomb[5][6] = {{{"KOKTELOK", "ALAPANYAGOK", "AUTO-KOK", "OPERACIOK", "KILEPES"}, {"LISTAZAS",
"HOZZADAS", "TORLES", "VISSZA"}, {"LISTAZAS", "HOZZADAS", "TORLES", "VISSZA"}, {"KERES", "VISSZA"},
{"BEOLVASAS", "MENTES", "VISSZA"}}};

/* Melyik menüpontra ment az előző és a mostani választásban? */
int elozo=0;
int melyik = 0;

```

```

while(true){
    melyik=navigation(tomb[melyik]);

    if(melyik==-1) /* Kilépés gombra nyomott. */
        break;
    elozo=melyik;

    melyik=navigation(tomb[melyik]); /* Almenü megnyitása a megfelelő menüpontokkal */

    switch(elozo){
        case 1:
            switch(melyik){
                case 1: /* KOKTEL LISTAZAS*/
                    std::cout<<"A taroloban "<<tarolo.db<<" koktel van."<<std::endl;
                    for (size_t i = 0; i < tarolo.db; ++i) {
                        std::cout << tarolo[i]; /*Az operátor szépen megjeleníti a koktél adatait*/
                    }
                    break;
                case 2: /*KOKTEL HOZZADAS*/
                    ...

```


5. BŐVÍTHETŐSÉG

Egy objektumorientált tervezésben kulcsfontosságú kérdés, hogy hogyan lehet bővíteni a programot. Ennek a programnak az esetében nagyon egyszerűen lehetne módosítani, hogy tulajdonképpen milyen alapanyag típusai lehetnek egy kódtételnek.

A program három típussal dolgozik, de tulajdonképpen könnyen létre lehetne hozni még több alapanyag típust. Ha így tennénk, ezt az Alapanyag osztályból kéne analitikusan örököltetni, és az ősosztályban a virtuális getter függvényeket kéne létrehozni. Az alapanyag tárolót teljes mértékben lehetne újrahaznosítani, hogyha string típusú adatokat akarunk tárolni. Ha mást is szeretnénk tárolni nevén kívül akkor csak bővíteni kéne a kívánt adattagokkal.

A menü is könnyen bővíthető számtalan menüponttal, csak a menü 2 dimenziós tömbjét kell bővíteni, majd a főmenü függvényben a switchben megvalósítani őket.

6. TESZTELÉS

6.1 Gtest lite

A program definiálva van a main.cpp és a menurendszer.cpp-ben egy SIM nevű makró. Ha ez definiálva van a programnak a interaktív és felhasználóbarát részei inaktívak és a szabványos bemenet át lesz irányítva egy megadott standard_input.txt fájlra, a szabványos kimenet pedig egy ostreamstream-be van irányítva. Erre azért van szükség, mert a megadott txt fájlban egy előre megadott lehetséges bemenet esetén determinisztikus kimenet várható el, így ez be van táplálva a programnak.

A teljesen tökéletes kimenet, mint egy változó szerepel a programban, és a program lefutásának végén ezt a kettőt hasonlítja össze. Ha bármilyen része a programnak nem működik megfelelően, a teszt nem lesz sikeres.

Az említett tesztek egyébként a program menüjén végigmegy, és minden funkciót kipróbál, többször is néhányat.

Ha a Gtest valamiért nem működne megfelelően, egy C++ nyelven megvalósított rész ugyan ezt a tesztet megcsinálja, és ha sikeres volt a teszt, akkor ezt a szabványos kimenetre írja, ami immár vissza lett irányítva.

Szabványos kimenet

```
TELJES PROGRAM SIKERESEN FUTOTT LE!
```

6.3 Jporta Coverage

A tesztesetek szinte teljes lefutást eredményeznek, ami a Jportán nem fut le az az interaktív menürendszer, és az econio nem használt elemei. Az Alapanyag ősosztály virtuális getter függvényei rendeltetés szerűen nincsenek meghívva. Ezeket kivéve a coverage: 97%+

6.4 Memória használat

A programban számos memória kezelés van, ennek ellenőrzésére a memtrace könyvtárat használtam, a teljes teszt után garantáltan nincs memóriaszivárgás.

```
Sikeres memóriaszivárgás teszt!
```

7. ÖN REVÍZIÓ ÉS MÓDOSÍTÁSOK

A korábbi Specifikáció és Terv csatolva van.

A koktélok tárolására lehetne esetleg láncolt listát használni, így meg lehetne spórolni a jelenleg nem használt koktélok memóriáját, azonban ez a leges legrosszabb esetben is csak 480 byte, ami gyakorlatilag semmilyen számítógépnek nem jelent problémát, mindazonáltal megvalósítása és algoritmusok implementálása sokkal komplikáltabb lenne, fölöslegesen.

Az econio helyett eleinte a System('Clear') és SetWindowAttribute volt használva, azonban probléma akadt a platformfüggőséggel, illetve egy rendszerfolyamat indítása a képernyő törlésére kimondottan rossz hatékonyságot okoz, ehelyett az econio könyvtár lett használva.

A megvalósítás során eleinte tisztán virtuális getter függvények voltak használva, azonban ezek le lettek cserélve nem tisztán virtuálisakra, aminek két oka van. Az egyik, hogy a leszármazott osztályokban csak a szükséges függvényeket kell megírni, így a kéretlen működés és a kód duplikáció is elkerülhető. Ezen felül az alaposztályra is jól működnek a függvények.

Eleinte a menürendszer menüindexeléssel működött, ez azt jelenti, hogy a felhasználó által kívánt menüpont számát kellett bevinni a billentyűzeten, azonban ez le lett cserélve egy véleményem szerint esztétikusabb menürendszerre.

8. FELHASZNÁLT KÖNYVTÁR

8.1 econio

A programban az egyetlen felhasznált külső könyvtár az econio. Két fájlt tartalmaz: econio.cpp és econio.h. Amiért szükség van rá az a cross platform képernyő törlés, és a szintén cross-platform azonnali billentyűzetbevitel. Függvények:

int econio_getchar(): Meghíváskor a billentyűzeten leütött billentyű kódját adja vissza egy számként, ezt felhasználva képes a felhasználó a menüben fel és le mozogni a „s” és „w” billentyűkkel, anélkül, hogy mindig entert kelljen nyomni. Megvalósítása meglehetősen bonyolult!

econio_clearscreen(): Korábban említett módon minden platformon törlni a konzolon lévő tartalmat. (Új process indítása nélkül).

9. A PROGRAM RÉSZEI

A main-ben történik a tesztelés, vagy a főmenü meghívása.

Az alapanyag-ot majdnem minden fájl látja, tartalmazza az alapanyag ősosztály-t és a származtatott osztályok megvalósítását.

A Koktél az egy koktél működését valósítja meg, a koktel_tarolo látja és a menurendszer.

A koktel_tarolo fájlban történik a tároló osztályok megvalósítása. A koktel_tarolóban koktélok, az alapanyag tárolóban alapanyagok nevei szerepelnek. A menurendszer fér hozzá.

A menürendszer fájlban van a programnak lefutásának alapja. A megjelenítést segítő függvények mellett itt található a főmenü. Csak a main látja és használja.

Az econio a korábban említett módon a konzolos működést segíti elő, menürendszerben van használva.

File	blank	comment	code
menurendszer.cpp	52	37	265
econio.cpp	69	19	249
koktel_tarolo.cpp	43	47	193
econio.h	32	44	80
alapanyag.h	27	27	77
koktel.cpp	26	17	67
koktel.h	30	19	47
koktel_tarolo.h	27	23	42
main.cpp	29	4	33
alapanyag.cpp	5	6	18
menurendszer.h	8	13	10
SUM:	348	256	1081